A machine learning enabled long-term performance evaluation framework for NoCs

Jie Hou
Chair of Embedded Systems
University of Stuttgart
Stuttgart, Germany
jie.hou@informatik.uni-stuttgart.de

Qi Han
Institute for Visualization and Interactive Systems
University of Stuttgart
Stuttgart, Germany
qi.han@vis.uni-stuttgart.de

Martin Radetzki
Chair of Embedded Systems
University of Stuttgart
Stuttgart, Germany
martin.radetzki@informatik.uni-stuttgart.de

***Abstract*—The rapidly increasing transistor density enables the evolution of many-core on-chip systems. Networks-on-Chips (NoCs) are the preferred communication infrastructure for such systems. Technology scaling increases the susceptibility to failures in the NoC's components. However, such a NoC can still operate at the cost of performance degradation. Therefore, it is not sufficient to analyze the performance and reliability of a NoC separately. In this paper, we propose a machine learning enabled performability evaluation framework to treat both aspects together. It applies Markov reward models. In addition, it leverages machine learning techniques to obtain different performance metrics under consideration of faulty routers and various simulation parameters quickly, which is a challenging task in an analytical manner. Moreover, we use a mesh-based NoC to demonstrate our methodology. Long-term performances of mesh 8×8 under XY and fault-tolerant negative-first routing algorithms are evaluated.**

***Index Terms*—Long-term performance; machine learning; supervised learning; fault-tolerant routing; Markov models; Markov reward models; NoCs; Mesh**

I. Introduction

Many-core systems-on-chip (SoCs) become more and more popular as they meet the continuously increasing demands for higher on-chip computing power. NoCs have emerged as an interconnection network infrastructure for such systems, as they overcome the performance and scalability issues in comparison with the traditional shared bus structure. In order to design a NoC of good performance, a number of factors such as topologies, routing algorithms, flow controls, etc. need to be considered [1].

As technology scaling develops, the transistor size continues to decrease. On the one hand, it enables the evolution of larger-size NoCs. On the other hand, it increases the susceptibility of NoCs components such as routers or links to suffer from faults [2]. Regarding their duration, these faults are classified into three types: permanent, transient and intermittent faults. A permanent fault will not disappear once it occurs. A transient fault is one that results in a defect in a component for some time, and the appeared defect goes away after that time and the component works completely well. In comparison with transient faults, an intermittent fault never entirely goes away, and it oscillates between being quiescent and active [3]. However, a NoC can still operate with degraded performance in the presence of faults. Therefore, such a NoC is classified as a fault-tolerant system.

Fault-tolerant systems that operate with faults and repairs can be treated as a Markov model [4]. Performability of such systems can be evaluated by means of Markov reward models [5]. In this work, we assume that routers suffer from transient faults, the failure rates and repair rates follow an exponential distribution. Consequently, such NoCs can be modeled as a continuous-time Markov chain (CTMC). With the help of Markov reward models, we can evaluate the performability of such NoCs. In other words, we evaluate the performance and reliability of a NoC jointly. In this context, reliability is the probability that the evaluated NoC has been operational during a specified time interval.

The remainder of this paper is structured as follows. Section II discusses related work relevant to the performability analysis of NoCs and machine learning methods used in various aspects of NoCs. Section III introduces the relevant preliminaries in the context of this paper. Section IV details our machine learning enabled framework to evaluate long-term performance of NoCs. Evaluation results are presented and discussed in Section V. Section VI concludes our work.

II. Related work

A. Performability analysis of NoCs

In the NoC literature, performability research can be classified into two categories. Works from the first category focus on various coding algorithms and error control schemes. For example, authors in [6] analyzed three error control schemes with joint consideration of performance, reliability, and energy consumption. They defined interconnect-performability as the probability to transmit a certain number of useful bits through a link of a NoC in the presence of noise for a specified time interval. In [7], authors proposed the performability of a network topology based on the definition of interconnect-performability from [6], connectivity matrix and traffic distribution matrix. The traffic distribution matrix specified the number of packets needs to be transmitted for each end-to-end communication flow. The connectivity matrix provided the number of links needs to be traversed by a packet from a source node to a destination node. Performabilities of several topologies were

978-1-7281-4882-3/19/$31.00 ©2019 IEEE
DOI 10.1109/MCSoC.2019.00031

analyzed by means of the proposed methodology. Authors in [8] defined performability as the probability of transmitting a correct flit. Different coding algorithms were analyzed with respect to the tradeoff between performability and energy. However, these works dealt with only the physical layer. The works from the second category used the classical performability definition to evaluate performabilities of NoCs. E.g.: in [9], the authors analyzed the performability of mesh-based NoCs using Markov reward model. However, they only evaluated the XY routing algorithm by means of communication time metric. They defined communication time as the sum of latencies of all used communication rounds for successfully delivering the specified number of packets. A communication round is a set of end-to-end communication flows, in which each flow sends only one packet and all flows send packets at the same time.

B. Machine learning methods used in various aspects of NoCs

Machine learning techniques have been applied in the different aspects of NoCs, e.g.: predicting traffic flow latency, reducing power consumption and exploring design space. In [10], authors developed a framework based on the kernel-based support vector regression to predict the channel average waiting time and the traffic flow latency. Usually, NoC latency models are based on the classical queueing theory. In order to model a buffer in a router as an $M/M/1$, $M/G/1/N$ or $G/G/1$ queue, following assumptions need to be made: 1) packet service time in the router is exponentially distributed, 2) packet length satisfies an exponential distribution. However, their proposed framework does not need to consider such assumptions. In [11], authors proposed learning enabled sleepy storage links and routers in NoCs to reduce both static and dynamic power consumption by power-gating the links and routers at low network utilization. They used the decision tree technique to make two predictions: predicting the link utilization to find out when to power-gate the channels and routers and predicting traffic load for changing the direction of the link. In [12], authors used a machine learning approach to intelligently explore the design space to optimize the placement of planar and vertical communication links. On average, the optimization achieved by them shows 35% energy-delay-product improvement.

In this paper, we propose a machine learning enabled framework to evaluate the performability of NoCs. Our main contributions are as follows:

1) use mesh-based NoCs to demonstrate our proposed machine learning enabled performability evaluation framework.
2) propose a methodology to obtain different performance metrics of NoCs based on machine learning techniques.
3) evaluate long-term performance of different routing algorithms: XY and fault-tolerant negative-first (FTNF).

III. Preliminaries

A. Markov reward model

One of the commonly used tools for performability analysis is Markov reward model. Formally, it contains two parts: a CTMC and a reward function. It extends a CTMC by assigning a reward to each state. A system operating with faults and repairs can be converted into a CTMC. A CTMC is described by a generator matrix denoted by Q, which contains information about states and transitions between states [4]. A reward r assigned to a state denotes a performance level provided by the system while it is in that state [5], [13].

Usually, a CTMC could be used for studying the dynamics of a system. Many techniques and algorithms have been developed to solve long-term steady-state distribution, transient state probabilities and occupancy times of Markov chains. Expected long-term reward rate and expected transient reward rate can be used for long-term and transient analysis of a system [5], [13]. In this paper we focus on the expected long-term reward rate denoted by Υ , which is defined as:

$$\Upsilon = \sum_{i \in \Omega} \pi_i r_i \tag{1}$$

where π_i means the long-term steady-state probability of residing in state i. π is obtained by solving the following equation:

$$\pi Q = \overrightarrow{0}, \ \sum_{i \in \Omega} \pi_i = 1 \tag{2}$$

By interpreting a reward rate as a performance level, the long-term performance level can be computed by Equation (1).

B. Supervised learning

There are three major classes of machine learning techniques: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the training dataset is a collection of labeled pairs $(x_i, y_i)_{i=1}^{N}$, where N denotes the total number of training samples, x_i is called a feature vector and y_i is the label of x_i. The goal of a supervised learning algorithm is to use the training dataset to find a function that determines which label should be given to a new feature vector. Supervised learning is divided into two major application types: classification and regression [14]. In classification problems, the goal is to predict a class label, which is a discrete value. The Spam filter is a good example of classification. Its input is an email, and its output is whether the provided email is Spam or not. In regression tasks, the goal is to predict a continuous number. E.g. predicting the price of a car given a set of features such as age, brand, mileage, etc [15].

C. Fault-tolerant routing

A routing algorithm defines a path taken by a packet between its source and destination. The path is determined within each intermediate router. In this paper, we focus on two routing algorithms: XY routing and FTNF routing. XY routing is deterministic, deadlock-free and easily implemented, but it cannot bypass faults. In the presence of faults, a fault-tolerant

routing algorithm introduces adaptivity to bypass them. The introduced adaptivity may result in deadlock. A well-known deadlock avoidance technique for the mesh-based NoC is the turn model [16]. The commonly used turn models are west-first, north-last, negative-first and odd-even. FTNF routing is based on negative-first turn model, and it was first presented in [17]. In the mesh topology, west and south are defined as negative directions, north and east denote positive directions. The negative-first turn model specifies that a packet traveling in the positive direction is forbidden to turn in negative directions. Table I describes briefly how we implemented the FTNF routing algorithm.

TABLE I
IMPLEMENTATION OF THE FTNF ROUTING ALGORITHM

	Incoming direction				
Destination router to current router	Local	N	E	S	W
NE	EN^*	ES	N	EN^*	EN^*
E (1 current router on south edge; 2 others)	EN^1 SE^2	SE	$-$	E	E
N (1 current router on west edge; 2 others)	NE^1 WN^2	$-$	NE^1 WN^2	N	N
NW (1 current router on south edge; 2 others)	WN^1 WS^2	WS	WS	WN	NE
W (1 current router on south edge; 2 others)	WN^1 WS^2	WS	WN^1 WS^2	WN	$-$
SE (1 current router on west edge; 2 others)	SE^1 SWE^2	SE^1 SWE^2	SW	EN	SE
S (1 current router on west edge; 2 others)	SE^1 SW^2	SE^1 SW^2	SW	$-$	SE
SW	WS^*	WS^*	WS^*	WN	SE

$*$: Select the direction that could provide higher path diversity

IV. THE MACHINE LEARNING ENABLED LONG-TERM PERFORMANCE EVALUATION FRAMEWORK

Our machine learning enabled long-term performance evaluation framework contains two parts: 1) model a NoC as a CTMC, 2) obtain performance metrics for validate states.

A. Markov model of a mesh-based NoC

In this paper, we use mesh topology to demonstrate how to model a NoC as a CTMC. We assume that routers suffer from transient faults. Two types of repair actions are modeled: local repair and global repair. The local repair can be thought of the disappearance of a transient fault. The disappearance requires some time. Moreover, our model of local repairs assumes that each router has an independent repair process. The global repair can be thought of like a reset.

Routers at different positions in a mesh-based NoC affect its performance differently. When we model it as a CTMC, we take into account positions of routers. Figure 1 illustrates our Markov model of a mesh-based NoC. As can be seen, we model a mesh-based NoC as a CTMC from the perspective of topology. As mesh has three different vertex degrees, vertexes are classified into three groups. In this context, a vertex denotes a router. The routers from $Group_1$ locate at four corners. The maximum size of $Group_1$ is 4. Routers from outer edges belong to $Group_2$. The other routers inside a mesh are classified as $Group_3$. We assume that routers from each group have the same failure and repair rates. The failure and repair rates related to routers from $Group_1$ are λ_1 and μ_1. Similarly, for routers from $Group_2$ and $Group_3$ they are λ_2, μ_2 and λ_3, μ_3 respectively. A triple (i, j, k) represents a state. It indicates the number of functional routers from the groups: $Group_1$, $Group_2$, and $Group_3$. A red line with an arrow represents a failure, and a black line with an arrow means a local repair. The global repair is denoted by the blue line with a weight of μ. $Phase\_x$ contains such states, whose total number of faulty routers is equal to x. When a certain number of routers are faulty, the NoC's performance may become so bad that it is not suitable for fulfilling communication tasks. Therefore, the total allowed number of faulty routers needs to be limited. It is defined as the fault limit and denoted by n. We briefly describe the state transitions in the model as follows:

- When a router from a group suffers from faults, a state moves forward to a next state. E.g. for the situation that a fault in $Group_3$ happens, it makes the model move from state (i, j, k) to state $(i, j, k-1)$ with rate of $k\lambda_3$.
- When a repair for a faulty router from a group is complete, a state moves backward to a previous state in case that the state is not a failure state. E.g. in state $(2, j, k)$ two routers from $Group_1$ are faulty. As each router has its own independent repair process, this state goes back state $(3, j, k)$ with a transition rate of $2\mu_1$. If the state is a failure state, the global repair will restore the NoC to its initial state.

B. The methodology to obtain performance metrics based on machine learning

In this work, two performance metrics are used: fault resilience (F) and communication time (CT). Fault resilience is the ratio of successfully delivered packets to the number of totally injected packets. It is an indicator of how reliable a NoC is in terms of connected paths. Its expression is as follows:

$$F = \frac{number\ of\ successfully\ delivered\ packets}{number\ of\ totally\ injected\ packets} \tag{3}$$

Communication time is the time a NoC needs for successfully delivering a specified number of packets under a given injection rate. Its expression is as follows:

$$\sum_{t=1}^{CT} \delta(t) >= N, where \sum_{t=1}^{CT-1} \delta(t) < N \tag{4}$$

where $\delta(t)$ represents the number of successfully delivered packets at simulation time t and N is the specified number of packets.

With consideration of faulty routers and other simulation parameters: packet injection rates, buffer depths, and routing algorithms, it is a challenging task to compute fault resilience and communication time in an analytical manner. In our work, we leverage machine learning techniques to predict fault resilience and communication time of NoCs under the mentioned

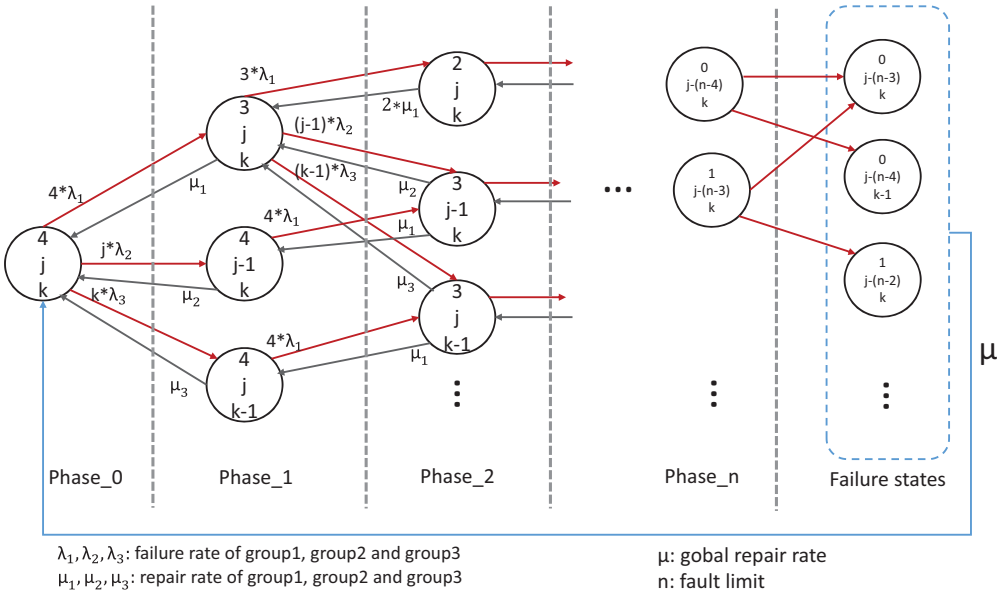

Fig. 1. Markov model of a mesh-based NoC

parameters. Figure 2 depicts an overview of our used methodology. It consists of three parts: datasets generation, finding suitable models as well as evaluating them, and applying them to compute performance metrics for performability analysis. Datasets are simulated results based on different configuration parameters. We use a cycle-accurate NoC simulator named Noxim for simulations [18]. It was implemented in $SystemC$. Parameters such as mesh size, buffer depth, packet injection rate, traffic pattern, routing algorithms etc. can be configured. We split a dataset into two sets: a training dataset with 80% of the total samples and a test dataset with 20% of the total samples. We use a training dataset to select a regression model, and the corresponding test dataset is used to evaluate the selected model.

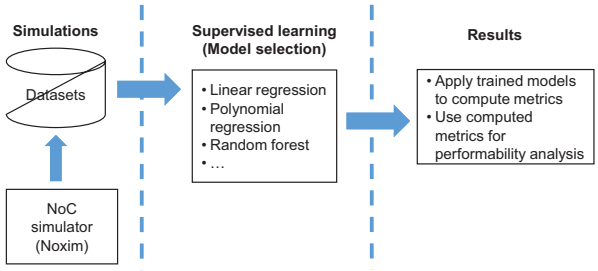

Fig. 2. Overview of the methodology to predict NoC's performance

1) Datasets generation workflow: In order to generate the datasets for training, we extended Noxim with fault injection, and the FTNF routing algorithm was implemented in it. Figure 3 shows the workflow of generating training data. It consists of following components: configuration file, datasets generator, the Noxim simulator and files storing generated datasets.

We detail how our program named datasets generator (DG) works in Algorithm 1, where:

- C: configuration parameters
- R: set of routing algorithms (XY and FTNF) specified in configuration
- P: set of packet injection rates specified in configuration
- T: set of performance metrics specified in configuration
- S: set of valid states from its Markov model
- Θ: set of generated fault combinations

The simulation parameters are stored in a configuration file. DG first loads it and parses the specified simulation parameters. Then, it generates a set of valid states based on our proposed Markov model IV-A ($Lines\ 2-6$). For each state, it generates a set of fault combinations. In our work, the maximum number of fault combinations generated for each state is 100 ($Line\ 8$). For each fault combination, different simulation commands are generated based on injection rates, routing algorithms and performance metrics. Before starting the simulation, the generated fault combination is injected into Noxim first. Then the DG waits until the simulation finishes. Once the simulation result is available, it appends the result to the corresponding dataset file ($Lines\ 9-18$).

In this paper, we use mesh 8×8 as an example to demonstrate our methodology. The maximum number of faulty routers is set as 7. Research papers [19], [20], [21] have shown that the NoC will eventually saturate with more and more

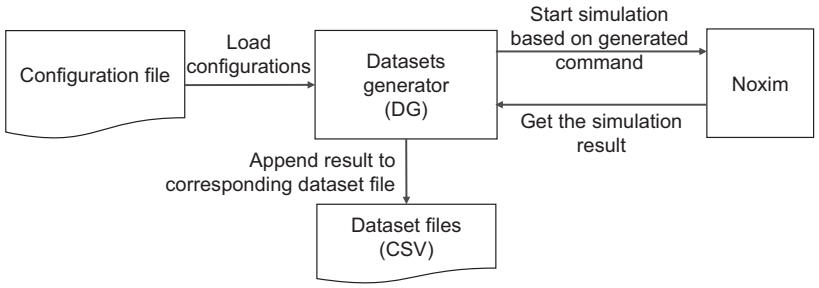

Fig. 3. Workflow of datasets generation

Algorithm 1 Procedure of generating datasets

```
1: procedure DatasetsGenerator(configFile)
2:     C ← loadConfiguration(configFile)
3:     R ← getRoutings(C)
4:     P ← getPacketInjectionRates(C)
5:     T ← getMetrics(C)
6:     S ← generateStates(C)
7:     for each s ∈ S do
8:         Θ ← generateFaultCombs(s, 100)
9:         for each θ ∈ Θ do
10:            injectFaults(θ)
11:            for each r ∈ R do
12:                for each p ∈ P do
13:                    for each t ∈ T do
14:                        command ←
15:                        generateCommand(r, p, t)
16:                        startSimulation(command)
17:                        result ← readResult()
18:                        appendResult(result, θ, p, r, t)
19:                    end for
20:                end for
21:            end for
22:        end for
23:    end for
24: end procedure
```

packets injected into the network. After a NoC is saturated, it behaves abnormally. It is not worth studying its performance anymore. In order to generate the valid datasets from simulation, we need to find out the saturating packet injection rate first. Our procedure for determining the saturating packet injection rate contains two steps: 1) obtain a set of global average delays under different packet injection rates, 2) find out the packet injection rate at which the global average delay increases drastically. After carrying out experiments, we find the saturating packet injection rates for XY and FTNF routings on the mesh 8×8 in the presence of 7 faults are 0.03 and 0.013 packet/cycle/node with following simulation parameters:

- Packet size: 5 flits
- Buffer depth: 6 flits
- Traffic pattern: uniform random

The above listed simulation parameters are also used for generating datasets, and we choose the packet injection rate in a range from 0.002 to 0.012 packet/cycle/node, as mesh 8×8 will not become saturated under both XY and FTNF algorithms in this range. For the metric of communication time defined by Equation (4), the specified number of packets that need to be successfully delivered is set to 5000. For the metric of fault resilience, the simulation time is set as 10000 cycles. For each metric under each routing, a dataset is generated. Finally, we obtained four datasets that are publicly available. [1]

2) Model selection: For each metric under each routing, a model needs to be selected. Three commonly used regression models are chosen as candidates: linear regression, polynomial regression, and random forest regressor. In each generated dataset, the number of features is 65: packet injection rate and statuses of 64 routers. In case that the degree of the polynomial regression is set to 2, the total number of transformed features is 2211. If the degree of the polynomial regression is set to 3, the total number of transformed features is 50116. It takes a lot of time to train such a model. Finally, we choose the polynomial regression of degree 2. Random forest regressor is an ensemble learning method. It consists of a certain number of decision trees and uses averaging to improve the predictive accuracy and control overfitting. The number of decision trees is set to 100 after conducting experiments.

We use the k-fold cross-validation technique to choose the regression model that fits data well. The algorithmic description of model selection is listed in Algorithm 2, where:

- M: set of candidate regression models
- D: a training dataset
- k: total number of cross-validation rounds
- Π: set of equally-sized subsets
- Δ: array stores average cross-validation score of each model

First a training dataset is equally split into k subsets ($Line$ 2). Then we perform totally k validation rounds for each candidate model. In each round, one of the k subsets is used as

[1] https://github.com/jiehou/noc_data

Algorithm 2 Procedure of model selection

1: **procedure** $ModelSelection(M, D, k)$
2: $\Pi \leftarrow partitionData(D, k)$ ▷ $\Pi = D_1, ..., D_k$
3: $\Delta \leftarrow \emptyset$
4: **for each** $m \in M$ **do**
5: $score \leftarrow \emptyset$
6: **for** $i = 1...k$ **do**
7: $trainModel(m, D \backslash \Pi[i])$
8: $score[i] \leftarrow computeValidationScore(m, \Pi[i])$
9: **end for**
10: $\Delta[m] \leftarrow \frac{1}{k} \sum_{i=1}^{k} score[i]$
11: **end for**
12: $return\ selectModel(\Delta[m])$
13: **end procedure**

the validation dataset and other $(k-1)$ subsets are for training the candidate model. The validation score used in our work is root mean squared error (RMSE). Finally, the validation scores of the k rounds are averaged and stored for the corresponding model ($Lines\ 4-11$). The model that achieves best average cross-validation score is selected ($Line\ 12$).

In the field of applied machine learning, k is normally selected as 5 or 10 in the k-fold cross-validation. In this paper, we set k to 5. We use a training dataset to perform model selection. 80% of the data from a training dataset is for training in each validation round, and 20% of the data from a training dataset is for validation.

TABLE II
CROSS-VALIDATION SCORES OF REGRESSION MODELS FOR COMMUNICATION TIME

Regression models	RMSE		Relative error	
	XY	$FTNF$	XY	$FTNF$
Linear regression	7838.26	5932.08	32.74%	31.21%
Polynomial regression (degree=2)	3714.69	2797.63	15.52%	14.72%
Random forest (num_trees=100)	2800.98	1389.24	11.70%	7.31%

TABLE III
CROSS-VALIDATION SCORES OF REGRESSION MODELS FOR FAULT RESILIENCE

Regression models	RMSE		Relative error	
	XY	$FTNF$	XY	$FTNF$
Linear regression	0.035	0.039	4.59%	4.14%
Polynomial regression (degree=2)	0.009	0.013	1.18%	1.38%
Random forest (num_trees=100)	0.007	0.004	0.92%	0.42%

The cross-validation scores and relative errors are listed in the Table II and Table III. The relative error is defined as the ratio of the value of RMSE to the average value of its corresponding training dataset. Based on them, we select random forest regressor for predicting communication time and fault resilience under XY and FTNF routing algorithms, because it achieves best scores for both performance metrics.

V. EXPERIMENTS

A. Model evaluation

In this section, we detail how the selected model works on the test datasets for communication time and fault resilience. Initially, the datasets from simulations are randomly shuffled and split into training and test datasets. The training datasets are used to select models. The test datasets are saved and untouched. First we train the random forest regressor with 100 decision trees using the whole training dataset, then we evaluate the trained model on the corresponding untouched test dataset. In addition to the RMSE score, we define the accuracy score as follows:

$$Accuracy = 1 - \frac{1}{m} \sum_{i=1}^{m} (|\hat{y_i} - y_i|/y_i) \tag{5}$$

where $\hat{y_i}$ denotes the predicted value of the i^{th} data from a test dataset, y_i is the label value of the i^{th} data from a test dataset, and m represents the size of a test dataset.

TABLE IV
EVALUATION SCORES OF MODELS FOR COMMUNICATION TIME

Routing	RMSE	Accuracy	Training time (sec)
XY	2779.39	93.32%	86.07
FTNF	1319.93	95.93%	92.87

TABLE V
EVALUATION SCORES OF MODELS FOR FAULT RESILIENCE

Routing	RMSE	Accuracy	Training time (sec)
XY	0.0069	99.26%	72.68
FTNF	0.0038	99.70%	74.89

Table IV and Table V list the scores of trained random forest models for communication time and fault resilience under XY and FTNF algorithms. As can be seen, the accuracies of the trained models for predicting communication times under XY and FTNF routing algorithms are 93.32% and 95.93%. For fault resilience, the accuracies of the trained models for XY and FTNF algorithms are 99.26% and 99.70%.

The total time to get four datasets from simulations is 4245 minutes. The average time to get a performance metric from the Noxim simulator is 566 ms. With consideration of training time, the average time to get a performance metric under the proposed machine learning approach is 3.7 ms, which is $153\times$ faster than the Noxim simulator. However, the model can be trained only once and saved for future use. Without consideration of training time, the average time to compute a performance metric under the machine learning approach is 0.0827 ms. In this case, we get an average speedup of $6844\times$ comparing to the Noxim simulator in terms of execution time.

B. Long-term performance evaluation

In order to evaluate the long-term performance of NoCs, we implemented a program to generate the NoC's Markov model as defined in Section IV-A and compute each state's long-term steady-state probability. In this study, we set the fault limit as 10 percent of the total number of routers. In the state space of the Markov model of a mesh 8×8, there are totally 145 states. The number of valid states is 110.

A reward of zero is assigned to each failure state. For each valid state, a performance metric needs to be computed. The number of combinations of faulty routers in some states may be huge. Each combination affects NoC's performance differently. Therefore, we need to compute a performance metric for each combination. Totally, a lot of time is needed to compute performance metrics. Obviously, such an approach is not optimal. In this paper, the Monte Carlo method is used to compute the average performance metric for each state. In such a state that represents less than 10000 fault combinations, performance metric is figured out for each combination and finally averaged. For other states, their metrics are computed using the Monte Carlo method. Furthermore, the specified minimum sample size is 10000 in this case. We assume all routers have the same failure and repair rates. Following parameters are used in our evaluation:

- Failure rate: $0.0015\ h^{-1}$
- Global repair rate: $0.03\ h^{-1}$
- Accuracy of Monte Carlo method: 0.001

1) Long-term communication time: If a mesh runs in the fault-free state, it has minimum communication time. In order to display performances in a normalized way, the communication time of the fault-free state is defined as *base time*. The reward rate associated with an operational state (i, j, k) is defined as follows:

$$r_{(i,j,k)} = \frac{Base\ time}{Communication\ time\ of\ state\ (i,j,k)} \quad (6)$$

A zero reward rate is assigned to each non-operational state. Combining with each state's steady-state probability, we can get the expected long-term reward rate. The long-term communication time is defined as the base time divided by the expected long-term reward rate.

Long-term communication times from XY and FTNF algorithms under six different packet injection rates and the repair rate of $0.02\ h^{-1}$ are illustrated in Figure 4. As can be seen, the long-term communication time from the FTNF routing is less than that from the XY routing under the same injection rate. That is to say, the FTNF algorithm needs less time to successfully deliver a specified number of packets. E.g. the long-term communication time achieved by the FTNF routing at the packet injection rate of 0.01 packet/cycle/node is 2347 cycles less than that from the XY routing. When the packet injection rate is in the range from 0.002 to 0.012 packet/cycle/node, the mesh will not be fully utilized. In this case, the throughput rises as the packet injection rate increases. For the communication time, the number of packets that need to be successfully delivered is constant, therefore it decreases with increasing packet injection rate until saturation.

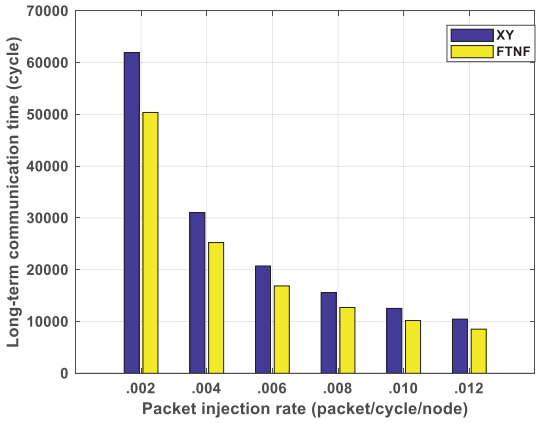

Fig. 4. Long-term communication times under different packet injection rates

2) Long-term fault resilience: Figure 5 illustrates long-term fault resiliences from both XY and FTNF routing algorithms under six different repair rates. As the repair rate increases, the long-term fault resiliences from XY and FTNF routing algorithms increase as well, because a NoC has larger residing probability in valid states under a higher repair rate. E.g.: long-term fault resilience from FTNF routing under the repair rate of $0.035\ h^{-1}$ is 0.9499, which is 0.1716 higher than that under the repair rate of $0.01\ h^{-1}$. Under the same repair rate, the long-term fault resilience from FTNF is greater than that from XY routing. In other words, FTNF routing is more resilient than XY routing from the perspective of long-term fault resilience. E.g. under the repair rate of $0.02\ h^{-1}$, the long-term fault resilience from FTNF routing is 15.2% more resilient than that from XY routing.

VI. Conclusion

This paper presents a machine learning enabled long-term evaluation framework for NoCs. First, we modeled a mesh-based NoC as a CTMC. Then, we presented our methodology based on machine learning techniques to compute two performance metrics: fault resilience and communication time. From evaluation results, we make the following conclusions:

1) By means of machine learning techniques, we are able to obtain metrics of fault resilience and communication time under consideration of a different number of faulty routers, packet injection rates, buffer depths, and routing algorithms. Under FTNF routing, we can predict fault resilience with the accuracy of 99.70% and communication time with the accuracy of 95.93%. Under XY routing, the accuracies of getting fault resilience and communication time are 99.26% and 93.32%. Of particular interest, the approach to computing a performance

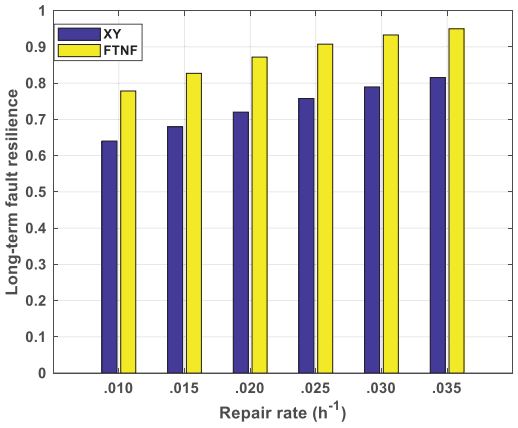

Fig. 5. Long-term fault resiliences under different repair rates

metric by means of pre-trained models has an average speedup of $6844\times$ comparing to the Noxim simulation in terms of execution time.

2) Before the NoC becomes saturated, the long-term communication time from FTNF routing is smaller than that from XY routing. Moreover, as the packet injection rate increases, the long-term communication time from XY and FTNF routing algorithms decrease until saturation.
3) As the repair rate increases, the long-term fault resiliences from XY and FTNF routing algorithms increase as well. Under the same repair rate, FTNF routing is more resilient than XY routing from the perspective of long-term fault resilience. E.g. under the repair rate of $0.02\ h^{-1}$, the long-term fault resilience from FTNF routing is 15.2% more resilient than that from XY routing.

REFERENCES

[1] N. E. Jerger and L.-S. Peh, "On-chip networks," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–141, 2009.

[2] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks-on-chip," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 8, 2013.

[3] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Elsevier, 2010.

[4] K. S. Trivedi, *Probability and statistics with reliability, queuing, and computer science applications*. Wiley Online Library, 1982, vol. 13.

[5] K. S. Trivedi, E. C. Andrade, and F. Machida, "Combining performance and availability analysis in practice." *Advances in Computers*, vol. 84, pp. 1–38, 2012.

[6] A. Ejlali, B. M. Al-Hashimi, P. Rosinger, S. G. Miremadi, and L. Benini, "Performability/energy tradeoff in error-control schemes for on-chip networks," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 1, pp. 1–14, 2010.

[7] H. Elmiligi, A. A. Morgan, M. W. El-Kharashi, and F. Gebali, "Improving networks-on-chip performability: A topology-based approach," *International Journal of Circuit Theory and Applications*, vol. 39, no. 6, pp. 557–572, 2011.

[8] A. A. Salem, M. A. A. El Ghany, and K. Hofmann, "Performability measurement of coding algorithms for network on chip," in *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 691–694.

[9] J. Hou and M. Radetzki, "Performability analysis of mesh-based nocs using markov reward model," in *Parallel, Distributed and Network-based Processing (PDP), 2018 26th Euromicro International Conference on*. IEEE, 2018, pp. 609–616.

[10] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "Svr-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 354–357.

[11] D. DiTomaso, A. Sikder, A. Kodi, and A. Louri, "Machine learning enabled power-aware network-on-chip design," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1354–1359.

[12] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty, "Optimizing 3d noc design for energy efficiency: A machine learning approach," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2015, pp. 705–712.

[13] B. R. Haverkort, "Markovian models for performance and dependability evaluation," in *School organized by the European Educational Forum*. Springer, 2000, pp. 38–83.

[14] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.

[15] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[16] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 278–287, 1992.

[17] ——, "Fault-tolerant wormhole routing in meshes," in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*. IEEE, 1993, pp. 240–249.

[18] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Improving energy efficiency in wireless network-on-chip architectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 1, p. 9, 2018.

[19] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550)*. IEEE, 2000, pp. 73–84.

[20] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip," *IEEE Transactions on Computers*, vol. 57, no. 6, pp. 809–820, 2008.

[21] M. Tang and X. Lin, "Injection level flow control for networks-on-chip (noc)." *J. Inf. Sci. Eng.*, vol. 27, no. 2, pp. 527–544, 2011.